The bridge to possible

Nexus-as-Code

Kickstart your automation with ACI

Daniel Schmidt, CX Architect

cisco Live!

BRKDCN-2673

Cisco Webex App

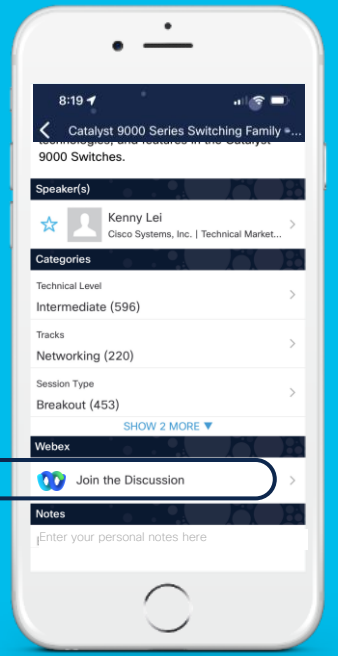
Questions?

Use Cisco Webex App to chat
with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install the Webex App or go directly to the Webex space
4. Enter messages/questions in the Webex space

Webex spaces will be moderated until February 24, 2023.



Agenda

- Infrastructure as Code
- Introduction to Nexus-as-Code
- Validation and Testing
- CI/CD Integration
- Scalability

Infrastructure as Code

Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

Infrastructure as Code (IaC) is the management of infrastructure in a descriptive model, using the same versioning as DevOps team uses for source code.

Infrastructure as Code (IaC) is the managing and provisioning of infrastructure through code instead of through manual processes.

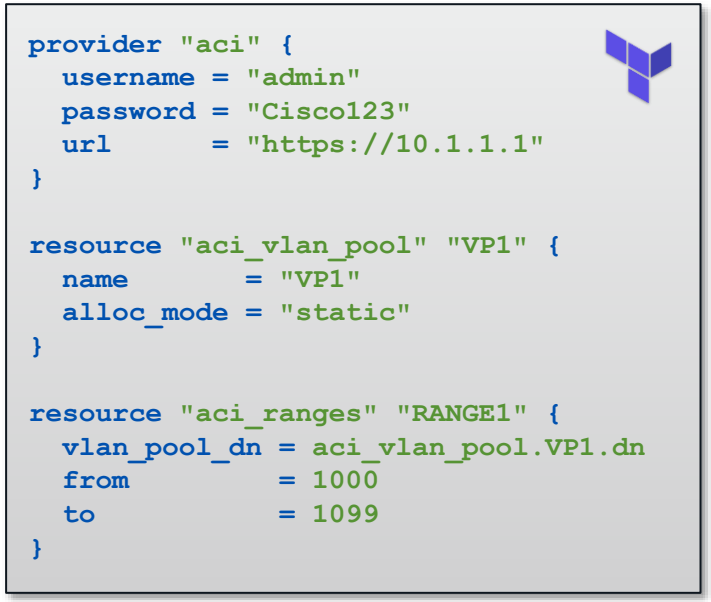
Practicing infrastructure as code means applying the same rigor of application code development to infrastructure provisioning. All configurations should be defined in a declarative way and stored in a source control system.

Infrastructure as Code is a process, not a single tool or application

Terraform Primer

Terraform is an Infrastructure Resources Manager

- Compose and combine infrastructure resources to build and maintain a desired state
- Plan and execution are distinct actions
- Manages all resources through APIs
- Terraform uses core and plugin components for basic functions and extensibility
- One of the most used IaC (Infrastructure-as-Code) tools to manage public Cloud and Datacenter assets
- HCL (Terraforms underlying configuration language) is the fastest growing language on GitHub in 2022 *



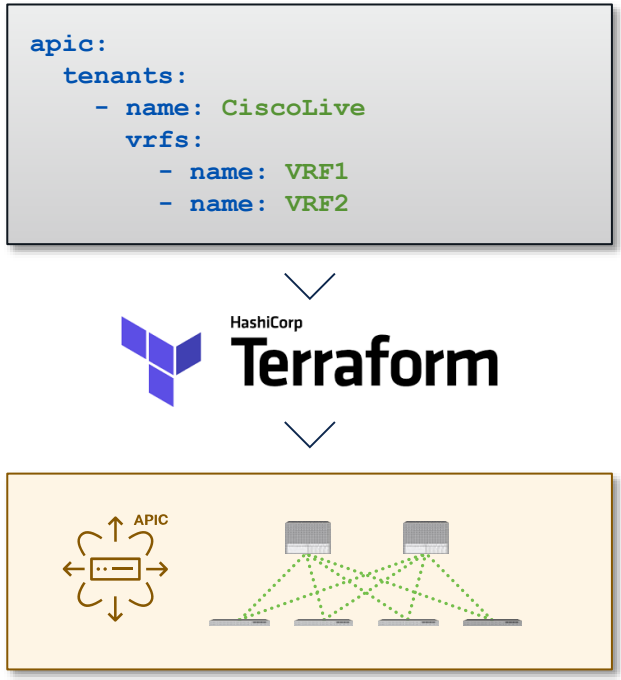
```
provider "aci" {
  username = "admin"
  password = "Cisco123"
  url      = "https://10.1.1.1"
}

resource "aci_vlan_pool" "VP1" {
  name       = "VP1"
  alloc_mode = "static"
}

resource "aci_ranges" "RANGE1" {
  vlan_pool_dn = aci_vlan_pool.VP1.dn
  from         = 1000
  to           = 1099
}
```

* https://octoverse.github.com/2022/top-programming-languages

Nexus-as-Code

https://cisco.com/go/nexusascode

- Nexus-as-Code aims to reduce time to value by lowering the barrier of entry to network orchestration through simplification, abstraction, and curated examples.
- It allows users to instantiate network fabrics in minutes using an easy to use, opinionated data model. It takes away the complexity of having to deal with references, dependencies or loops.
- Users can focus on describing the intended configuration while using a set of maintained and tested Terraform Modules without the need to understand the low-level ACI object model.



Comparison

Native Terraform

```
resource "aci_tenant" "tenant_CiscoLive" {
  name = "CiscoLive"
}

variable "vrfs" {
  default = {
    VRF1 = {
      name = "VRF1"
    },
    VRF2 = {
      name = "VRF2"
    }
  }
}

resource "aci_vrf" "vrfs" {
  for_each  = var.vrfs
  tenant_dn = aci_tenant.tenant_CiscoLive.id
  name      = each.value.name
}
```

Nexus-as-Code

```
apic:
  tenants:
    - name: CiscoLive
      vrfs:
        - name: VRF1
        - name: VRF2
```

Node Policies

- The data model is organized in a way that configurations are grouped around where the actual configuration (policy) is applied.
- All the configurations that are applied at the node level can be found under:
  apic -> node_policies -> nodes
- This includes configurations typically found in different places in the ACI object tree, like for example the OOB node management address, which is configured under the mgmt tenant.
- Consolidating all node level configurations in a single place eases maintenance, as for example we only have to update this single section when adding a new node.

```
apic:
  node_policies:
    nodes:
      - id: 101
        pod: 2
        role: leaf
        serial_number: FDO13026BEN
        name: leaf-101
        oob_address: 10.103.5.101/24
        oob_gateway: 10.103.5.254
        update_group: group-1
        fabric_policy_group: all-leafs
        access_policy_group: all-leafs

      - id: 1
        pod: 2
        role: apic
        oob_address: 10.103.5.1/24
        oob_gateway: 10.103.5.254
```

Access Policies

- A number of profiles and selectors can be auto-generated by providing a naming convention.
- There is no need to worry about any of the profiles and selectors as they will be added/deleted automatically according to the node and interface configuration.
- As nodes are added under *apic -> node_policies -> nodes* the corresponding profiles will be created automatically.
- Once interface configurations are added under *apic -> interface_policies -> nodes -> interfaces* the corresponding interface selectors will be created

```
apic:
  auto_generate_switch_pod_profiles: true

  interface_policies:
    nodes:
      - id: 101
        interfaces:
          - port: 1
            description: Linux Server 1
            policy_group: linux-servers
          - port: 2
            description: Linux Server 2
            policy_group: linux-servers
          - port: 47
            description: N7K Core
            policy_group: n7000-a
          - port: 48
            description: N7K Core
            policy_group: n7000-b
```

Simple Demo

https://github.com/netascode/nac-aci-simple-example

Separate Data from Code

In order to ease maintenance we separate data (variable definition) from logic (infrastructure declaration), where one can be updated independently from the other.

```yaml
apic:
  tenants:
    - name: CiscoLive
      vrfs:
        - name: CiscoLive
      bridge_domains:
        - name: vlan-100
          vrf: CiscoLive
      application_profiles:
        - name: dev
          endpoint_groups:
            - name: vlan-100
              bridge_domain: vlan-100
              physical_domains: ["l2"]
```

apic.yaml

```
locals {
  model = yamldecode(file("./apic.yaml"))
}

module "tenant" {
  source = "./modules/terraform-aci-tenant"

  for_each = toset(
    [for tenant in local.model.apic.tenants : tenant.name]
  )

  model       = local.model
  tenant_name = each.value
}
```

main.tf

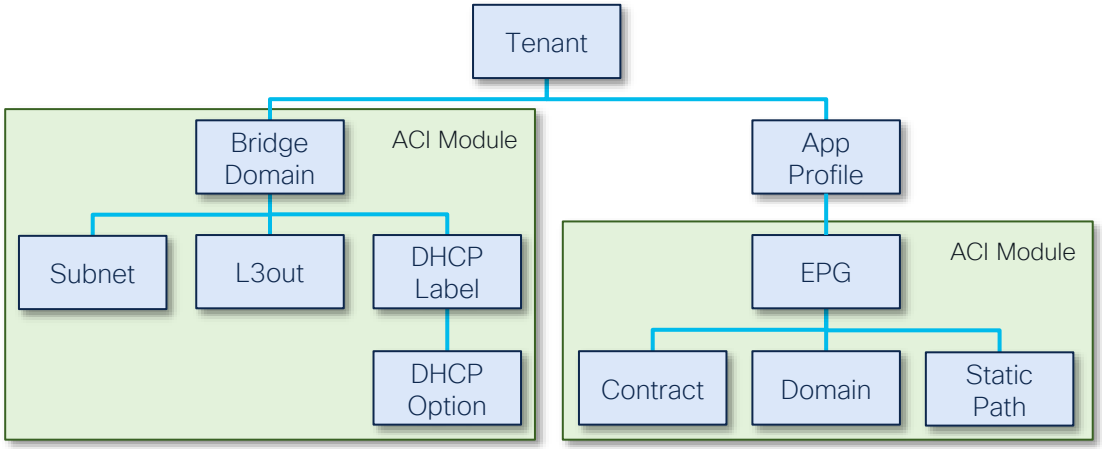
ACI Terraform Provider

- Nexus-as-Code heavily relies on the generic *aci_rest_managed* resource of the ACI Terraform provider.
- This fully-featured resource is able to manage any ACI object.
- The resource is not only capable of pushing a configuration but also reading its state and reconcile configuration drift.

```
resource "aci_rest_managed" "fvTenant" {
  dn = "uni/tn-EXAMPLE_TENANT"
  class_name = "fvTenant"

  content = {
    name = "EXAMPLE_TENANT"
    descr = "Example description"
  }

  child {
    rn = "ctx-VRF1"
    class_name = "fvCtx"
    content = {
      name = "VRF1"
    }
  }
}
```

ACI Modules

- Terraform Modules allow us to introduce a level of abstraction similar to functions in programming languages
- Where a Terraform resource typically represents a single ACI object, a Terraform module can represent a branch in the object tree



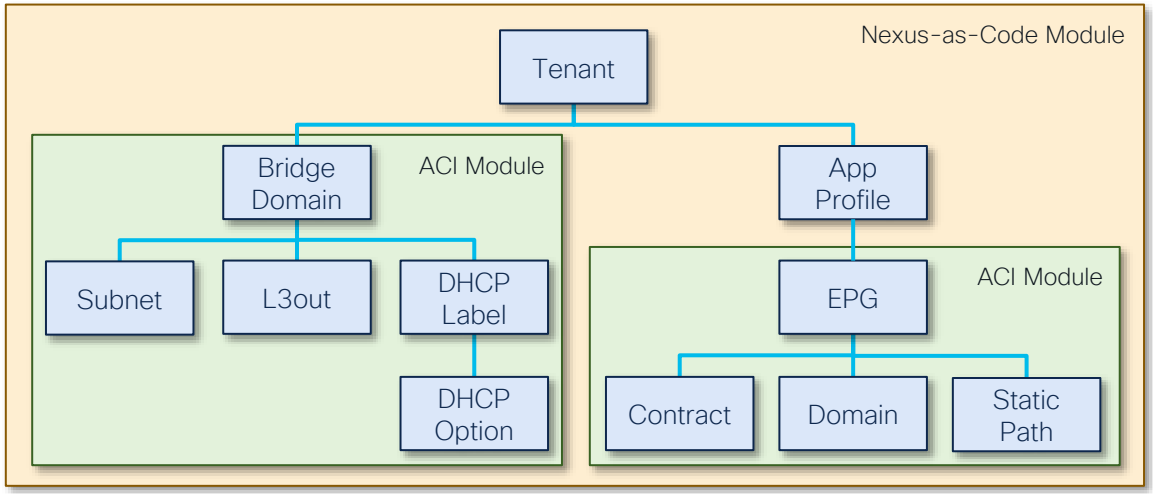
ACI Module Example

- Modules allow us to break a configuration into more manageable pieces which can be developed and tested independently
- Modules can be versioned and released independently
- Modules enable easier shareability and cut down on duplicate work as they can be shared with the wider community (Terraform Registry)
- Terraform recently introduced a testing experiment, which enables writing integration tests for modules directly in Terraform

```
module "aci_endpoint_group" {
  source  = "netascode/endpoint-group/aci"
  version = ">= 0.1.0"

  tenant               = "ABC"
  application_profile  = "AP1"
  name                 = "EPG1"
  bridge_domain        = "BD1"
  contract_consumers   = ["CON1"]
  physical_domains     = ["PHY1"]
  vmware_vmm_domains = [{
    name = "VMW1"
  }]
  static_ports = [{
    node_id = 101
    vlan = 123
    port = 10
  }]
}
```

Nexus-as-Code Modules

- Fabric Policies: Configurations applied at the fabric level (e.g., fabric BGP route reflectors)
- Access Policies: Configurations applied to external facing (downlink) interfaces (e.g., VLAN pools)
- Pod Policies: Configurations applied at the pod level (e.g., TEP pool addresses)
- Node Policies: Configurations applied at the node level (e.g., OOB node management address)
- Interface Policies: Configurations applied at the interface level (e.g., assigning interface policy groups to ports)
- Tenants: Configurations applied at the tenant level (e.g., VRFs and Bridge Domains)



YAML layout

- As different teams might be responsible for different parts of the infrastructure, it is of paramount importance to allow enough flexibility when defining and maintaining the ACI configuration.
- The configuration can be split into multiple YAML files each for example covering a specific logical section of the configuration.
- Nexus-as-Code does not dictate a specific schema, but instead allows for full flexibilty to divide the configuration as needed.

```
$ tree -L 2
.
├── data
│   ├── apic.yaml
│   ├── access_policies.yaml
│   ├── fabric_policies.yaml
│   ├── node_policies.yaml
│   ├── pod_policies.yaml
│   ├── node_1001.yaml
│   ├── node_101.yaml
│   ├── node_102.yaml
│   └── tenant_PROD.yaml
├── defaults
│   └── defaults.yaml
└── main.tf
```

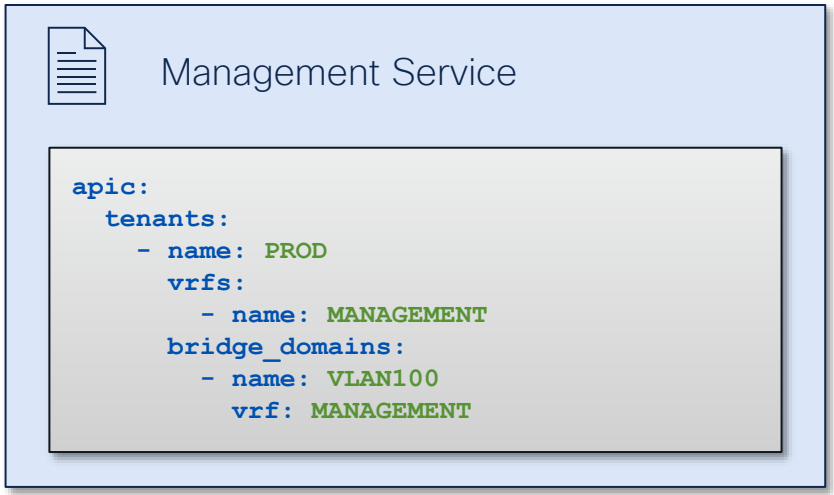
Merging YAML Files

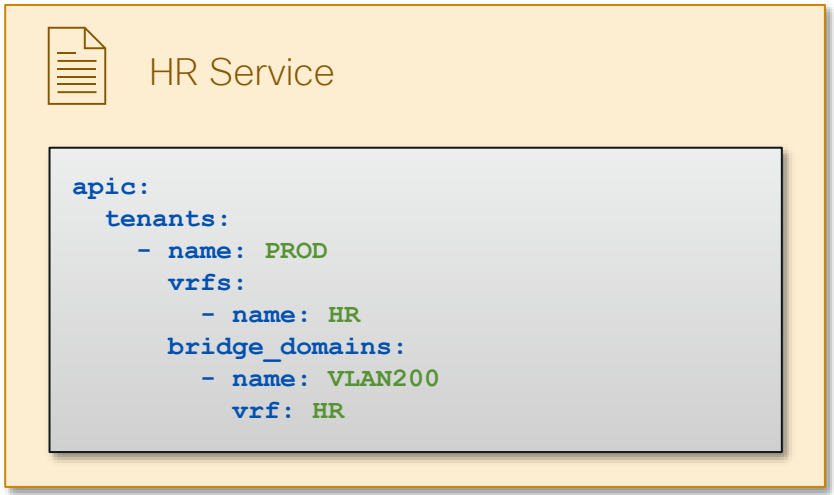
There is a dedicated Terraform Module available to merge the content of multiple YAML files into a single data structure which can then be provided to the Nexus-as-Code Terraform Modules.

```
module "merge" {
  source = "netascode/nac-merge/utils"
  version = "0.1.2"

  yaml_strings = [for file in fileset(path.module, "data/*.yaml") : file(file)]
}

module "access_policies" {
  source = "netascode/nac-access-policies/aci"
  version = "0.4.1"

  model = module.merge.model
}
```

Deep Merge YAML Content

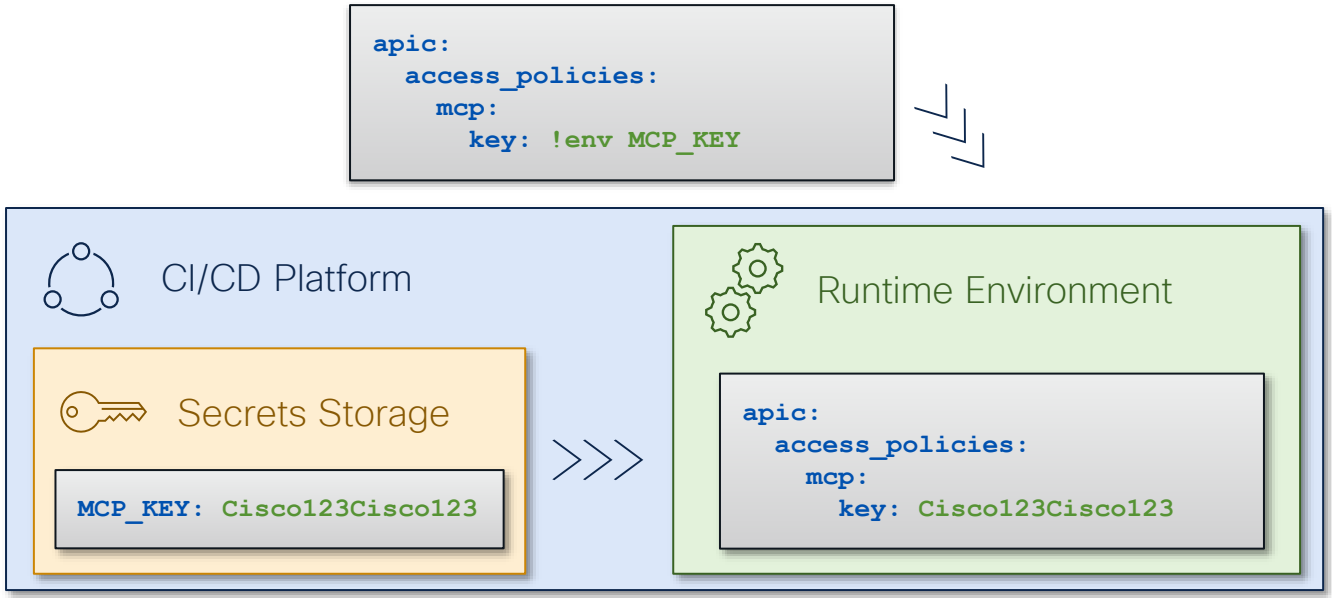
YAML files can be split at arbitrary points, meaning the 'nac-merge' Module will combine and deep merge the contents of YAML files, where data of two elements with the same keys will be combined. This for example enables splitting the configuration of a single tenant in two YAML files.





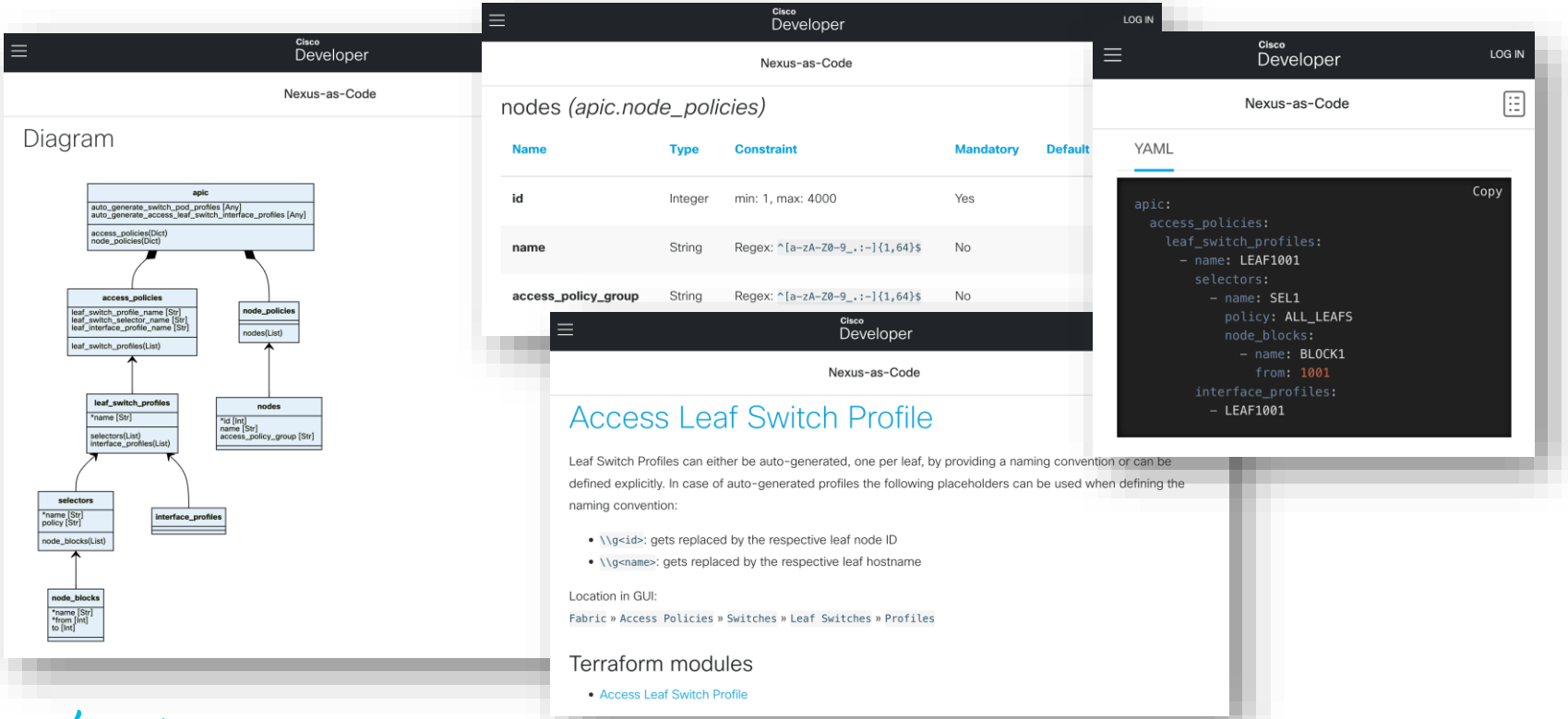
Secrets

The configuration might contain sensitive information that should not be stored in cleartext in the configuration. One common approach to handling secrets in the context of CI/CD Platforms is by injecting sensitive values as environment variables during runtime.



Data Model Documentation

https://cisco.com/go/nexusascode



Default Values

- Nexus-as-Code comes with pre-defined default values based on common best practices.
- In some cases, those default values might not be the best choice for a particular deployment and can be overwritten if needed.
- Appending suffixes to object names is a common practice that introduces room for human errors. Using default values, such suffixes can be defined once and then consistently appended to all objects of a specific type including its references.

```yaml
defaults:
  apic:
    tenants:
      bridge_domains:
        name_suffix: _bd
        unicast_routing: false
```

defaults.yaml

```yaml
apic:
  tenants:
    - name: CiscoLive
      bridge_domains:
        - name: vlan_101
        - name: vlan_102
        - name: vlan_103
```

tenants.yaml

Unmanaged Parent Objects

In some cases you might only want to manage objects within a container. The *managed* flag indicates if an object should be created/modified/deleted or is assumed to exist already and just acts a container for other objects.

Infrastructure Team manages Tenants

```
apic:
  tenants:
    - name: Dev
    - name: Stage
    - name: Prod
```

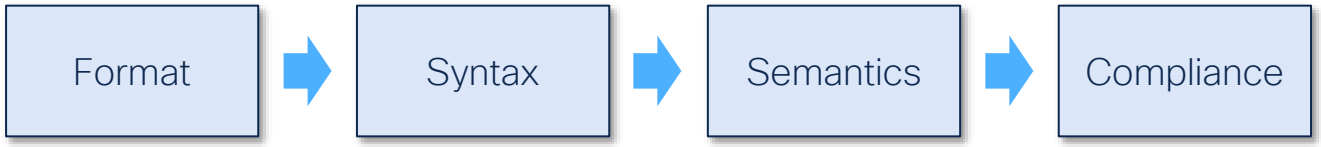
Developers manage Tenant Objects

```
apic:
  tenants:
    - name: Dev
      managed: false
      vrfs:
        - name: VRF1
        - name: VRF2
```

Pre-Change Validation

As the complexity of the configuration and the underlying data model increases automated validation before deploying anything in a production environment becomes a critical aspect.

Several tools can be used to ensure that the provided input data is valid, but also that common best practices and formatting guidelines are being followed.



Pre-Change Validation

A CLI tool to perform format, syntactic, semantic and compliance validation of Nexus-as-Code YAML files.

```
$ iac-validate -h
Usage: iac-validate [OPTIONS] [PATHS]...

  A CLI tool to perform syntactic and semantic validation of YAML files.

Options:
  --version               Show the version and exit.
  -v, --verbosity LVL     Either CRITICAL, ERROR, WARNING, INFO or DEBUG
  -s, --schema FILE       Path to schema file. (optional, default:
                          '.schema.yaml', env: IAC_VALIDATE_SCHEMA)
  -r, --rules DIRECTORY   Path to semantic rules. (optional, default:
                          '.rules/', env: IAC_VALIDATE_RULES)
  -o, --output FILE       Write merged content from YAML files to a new YAML
                          file. (optional, env: IAC_VALIDATE_OUTPUT)
  -h, --help              Show this message and exit.
```

Syntax Validation

- Native Terraform variable validation rules have limitations with complex and/or nested structures
- Tools like Yamale can be used to define the schema and validate YAML files against it
- The schema specifies the expected structure, input value types (String, Enum, IP, etc.) and additional constraints (eg. value ranges, regexes, etc.)

```
---
apic: include('apic', required=False)
---
apic:
  tenants: list(include('tenant'), required=False)

tenant:
  name: regex('^[a-zA-Z0-9_.:-]{1,64}$')
  vrfs: list(include('ten_vrf'), required=False)

ten_vrf:
  name: regex('^[a-zA-Z0-9_.:-]{1,64}$')
  alias: regex('^[a-zA-Z0-9_.:-]{1,64}$', required=False)
  data_plane_learning: bool(required=False)
  enforcement_direction: bool(required=False)
  contracts: include('ten_vrf_contracts', required=False)
```

Semantic Validation

Semantic validation is about verifying specific data model related constraints like referential integrity. It can be implemented using a rule based model like commonly done with linting tools. Examples are:

- Check uniqueness of key values (eg. Node IDs)
- Check references/relationships between objects (eg. Interface Policy Group referencing a CDP Policy)

```
Rule 101: Verify unique keys ['apic.node_policies.nodes.id - 102']
Rule 201: Verify references ['apic.node_policies.nodes.update_group - GROUP1']
Rule 205: Verify Access Spine Interface Policy Group references
['apic.interface_policies.nodes.interfaces.policy_group - SERVER1']
```

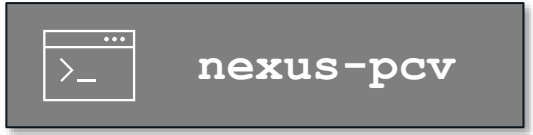
Compliance Validation

NDI Pre-Change Analysis

Nexus Dashboard Insights (NDI) is continuously pulling the entire policy, every configuration, and the network-wide state, along with the operator intent, and building from these comprehensive and mathematically accurate models of network behavior. It combines this with codified Cisco domain knowledge to generate “smart events” that pinpoint deviations from intent and offer remediation recommendations.

The Pre-Change Analysis feature can be used to assess the impact of a particular change before applying it to the infrastructure. This is done by applying the planned changes to the model and then analysing the impact.

NDI Pre-Change Analysis



A CLI tool to perform a pre-change analysis on Nexus Dashboard Insights or Network Assurance Engine. It can either work with provided JSON file(s) or a *terraform plan* output from a Nexus-as-Code project. It waits for the analysis to complete and evaluates the results.

```
$ nexus-pcv -h
Usage: nexus-pcv [OPTIONS]

  A CLI tool to perform a pre-change validation on Nexus Dashboard Insights or
  Network Assurance Engine.

Options:
  -i, --hostname-ip TEXT      NAE/ND hostname or IP (required, env:
                              PCV_HOSTNAME_IP).
  -u, --username TEXT         NAE/ND username (required, env: PCV_USERNAME).
  -p, --password TEXT         NAE/ND password (required, env: PCV_PASSWORD).
  -d, --domain TEXT           NAE/ND login domain (optional, default: 'Local',
                              env: PCV_DOMAIN).
  -g, --group TEXT            NAE assurance group name or NDI insights group
                              name (required, env: PCV_GROUP).
  -s, --site TEXT             NDI site or fabric name (optional, only required
                              for NDI, env: PCV_SITE).
```

Testing

There are certain aspects we can only verify after deployment like for example operational state. Various testing frameworks can be used for that, one example would be Robot Framework. Robot's language agnostic syntax with libraries like Requests and JSONLibrary can be used to write tests against REST APIs.

In combination with templating languages like Jinja we can render test cases dynamically based on the desired state.

Tests can typically be categorized in three groups:

- Configuration Tests: verify if the desired configuration is in place
- Health Tests: leverage the in-built APIC fault correlation to retrieve faults and health scores and compare them against thresholds and/or previous state
- Operational Tests: verify operational state according to input data, eg. BGP peering state

Testing

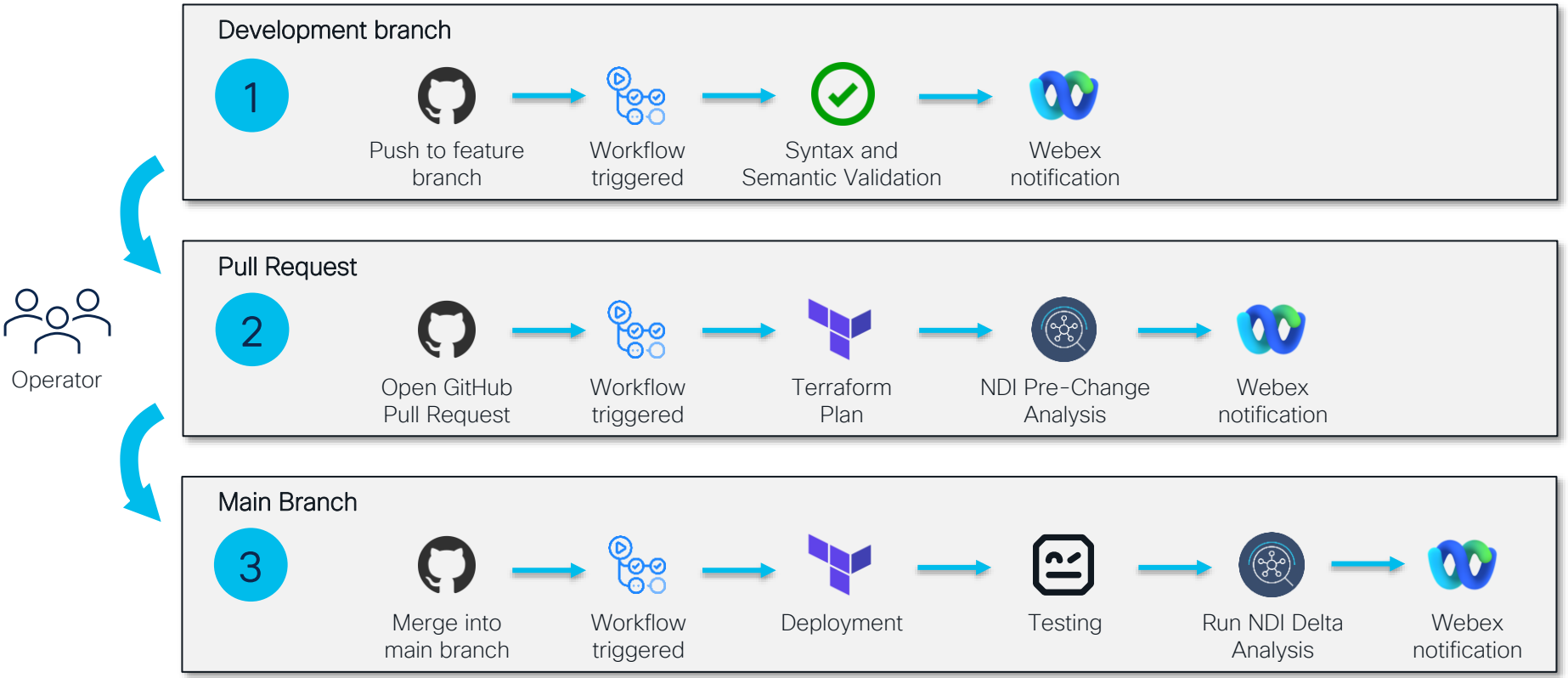
A CLI tool to render and execute Robot Framework tests using Jinja templating.

```
$ iac-test -h
Usage: iac-test [OPTIONS]

  A CLI tool to render and execute Robot Framework tests using Jinja
  templating.

Options:
  -d, --data PATH             Path to data YAML files. (env: IAC_TEST_DATA)
                              [required]
  -t, --templates DIRECTORY   Path to test templates. (env: IAC_TEST_TEMPLATES)
                              [required]
  -f, --filters DIRECTORY     Path to Jinja filters. (env: IAC_TEST_FILTERS)
  --tests DIRECTORY           Path to Jinja tests. (env: IAC_TEST_TESTS)
  -o, --output DIRECTORY      Path to output directory. (env: IAC_TEST_OUTPUT)
                              [required]
  -i, --include TEXT          Selects the test cases by tag (include). (env:
                              IAC_TEST_INCLUDE)
  -e, --exclude TEXT          Selects the test cases by tag (exclude). (env:
                              IAC_TEST_EXCLUDE)
  --render-only               Only render tests without executing them. (env:
                              IAC_TEST_RENDER_ONLY)
```

Robot/Jinja Example


```
*** Settings ***
Documentation    Verify Tenant Health
Suite Setup      Login APIC
Default Tags     apic    day2    health    tenants    non-critical
Resource         ../../apic_common.resource

*** Test Cases ***
{% for tenant in apic.tenants | default([]) %}
Verify Tenant {{ tenant.name }} Faults
    ${r}=    GET On Session    apic    /api/mo/uni/tn-{{ tenant.name }}/fltCnts.json
    ${critical}=    Get Value From Json    ${r.json()}    $..faultCountsWithDetails.attributes.crit
    Run Keyword If    ${critical} > 0    Run Keyword And Continue On Failure
    ...    Fail "{{ tenant.name }} has ${critical} critical faults"

Verify Tenant {{ tenant.name }} Health
    ${r}=    GET On Session    apic    /api/mo/uni/tn-{{ tenant.name }}/health.json
    ${health}=    Get Value From Json    ${r.json()}    $..healthInst.attributes.cur
    Run Keyword If    ${health} < 100    Run Keyword And Continue On Failure
    ...    Fail "{{ tenant.name }} health score: ${health}”

{% endfor %}
```

CI/CD Workflow Example



Scalability

By adding more and more objects to your configuration a few problems can arise:

- The Terraform state file becomes bigger and making changes with Terraform takes much longer.
- A single shared statefile is a risk. Making a change in a Development tenant could have implications to a Production tenant.
- No ability to run changes in parallel. Only one concurrent plan may run at any given time as the statefile is locked during the operation.
- With Nexus-as-Code, state can be split into multiple workspaces while retaining a single set of YAML files.

```
$ tree -L 2
.
├── data
│   ├── apic.yaml
│   ├── access_policies.yaml
│   ├── fabric_policies.yaml
│   ├── node_policies.yaml
│   ├── pod_policies.yaml
│   ├── node_1001.yaml
│   ├── node_101.yaml
│   ├── node_102.yaml
│   ├── tenant_PROD.yaml
│   ├── tenant_DEV.yaml
│   └── defaults.yaml
└── workspaces
    ├── tenant_PROD
    │   └── main.tf
    └── tenant_DEV
        └── main.tf
```

CI/CD Demo

https://github.com/netascode/BRKDCN-2673-Demo

cisco Live!

References

- Nexus-as-Code
  https://cisco.com/go/nexusascode
- Demo Repository
  https://github.com/netascode/BRKDCN-2673-Demo
- ACI Terraform Provider
  https://registry.terraform.io/providers/CiscoDevNet/aci/latest
- Pre-Change Validation Tool
  https://github.com/netascode/iac-validate
- Test Automation Tool
  https://github.com/netascode/iac-test
- NX-OS, IOS-XE, IOS-XR Terraform Providers
  https://registry.terraform.io/search/providers?q=netascode

Complete your Session Survey

- Please complete your session survey after each session. Your feedback is important.
- Complete a minimum of 4 session surveys and the Overall Conference survey (open from Thursday) to receive your Cisco Live t-shirt.

- All surveys can be taken in the Cisco Events Mobile App or by logging in to the Session Catalog and clicking the "Attendee Dashboard" at https://www.ciscolive.com/emea/learn/sessions/session-catalog.html

Continue Your Education

Visit the Cisco Showcase for related demos.

Book your one-on-one Meet the Engineer meeting.

Attend any of the related sessions at the DevNet, Capture the Flag, and Walk-in Labs zones.

Visit the On-Demand Library for more sessions at ciscolive.com/on-demand.


cisco
The bridge to possible
Thank you
cisco Live!


CISCO Live!
ALL IN